



## PIPELINE ACCELERATOR INCLUDING HARDWIRED-PIPELINE CIRCUITS IN COMMUNICATION VIA A BUS, AND RELATED SYSTEM AND METHOD

### CLAIM OF PRIORITY

- [1] This application claims priority to U.S. Provisional Application Serial No.  
5 60/422,503, filed on October 31, 2002, which is incorporated by reference.

### CROSS REFERENCE TO RELATED APPLICATIONS

- [2] This application is related to U.S. Publication Nos. 2004/0133763 entitled  
IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD,  
2004/0181621 entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING  
10 ARCHITECTURE AND RELATED SYSTEM AND METHOD, 2004/0136241 entitled  
PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND  
RELATED SYSTEM AND METHOD, and 2004/0170070 entitled PROGRAMMABLE  
CIRCUIT AND RELATED COMPUTING MACHINE AND METHOD, which have a  
common filing date and owner.

### 15 BACKGROUND

- [3] A common computing architecture for processing relatively large amounts  
of data in a relatively short period of time includes multiple interconnected processors  
that share the processing burden. By sharing the processing burden, these multiple  
processors can often process the data more quickly than a single processor can for a  
20 given clock frequency. For example, each of the processors can process a respective  
portion of the data or execute a respective portion of a processing algorithm.

- [4] FIG. 1 is a schematic block diagram of a conventional computing  
machine **10** having a multi-processor architecture. The machine **10** includes a master  
processor **12** and coprocessors **14<sub>1</sub> – 14<sub>n</sub>**, which communicate with each other and the  
25 master processor via a bus **16**, an input port **18** for receiving raw data from a remote  
device (not shown in FIG. 1), and an output port **20** for providing processed data to the  
remote source. The machine **10** also includes a memory **22** for the master  
processor **12**, respective memories **24<sub>1</sub> – 24<sub>n</sub>** for the coprocessors **14<sub>1</sub> – 14<sub>n</sub>**, and a

memory **26** that the master processor and coprocessors share via the bus **16**. The memory **22** serves as both a program and a working memory for the master processor **12**, and each memory  $24_1 - 24_n$  serves as both a program and a working memory for a respective coprocessor  $14_1 - 14_n$ . The shared memory **26** allows the master processor **12** and the coprocessors **14** to transfer data among themselves, and from/to the remote device via the ports **18** and **20**, respectively. The master processor **12** and the coprocessors **14** also receive a common clock signal that controls the speed at which the machine **10** processes the raw data.

[5] In general, the computing machine **10** effectively divides the processing of raw data among the master processor **12** and the coprocessors **14**. The remote source (not shown in **FIG. 1**) such as a sonar array loads the raw data via the port **18** into a section of the shared memory **26**, which acts as a first-in-first-out (FIFO) buffer (not shown) for the raw data. The master processor **12** retrieves the raw data from the memory **26** via the bus **16**, and then the master processor and the coprocessors **14** process the raw data, transferring data among themselves as necessary via the bus **16**. The master processor **12** loads the processed data into another FIFO buffer (not shown) defined in the shared memory **26**, and the remote source retrieves the processed data from this FIFO via the port **20**.

[6] In an example of operation, the computing machine **10** processes the raw data by sequentially performing  $n + 1$  respective operations on the raw data, where these operations together compose a processing algorithm such as a Fast Fourier Transform (FFT). More specifically, the machine **10** forms a data-processing pipeline from the master processor **12** and the coprocessors **14**. For a given frequency of the clock signal, such a pipeline often allows the machine **10** to process the raw data faster than a machine having only a single processor.

[7] After retrieving the raw data from the raw-data FIFO (not shown) in the memory **26**, the master processor **12** performs a first operation, such as a trigonometric function, on the raw data. This operation yields a first result, which the processor **12** stores in a first-result FIFO (not shown) defined within the memory **26**. Typically, the

processor **12** executes a program stored in the memory **22**, and performs the above-described actions under the control of the program. The processor **12** may also use the memory **22** as working memory to temporarily store data that the processor generates at intermediate intervals of the first operation.

5    **[8]**                Next, after retrieving the first result from the first-result FIFO (not shown) in the memory **26**, the coprocessor **14<sub>1</sub>** performs a second operation, such as a logarithmic function, on the first result. This second operation yields a second result, which the coprocessor **14<sub>1</sub>** stores in a second-result FIFO (not shown) defined within the memory **26**. Typically, the coprocessor **14<sub>1</sub>** executes a program stored in the  
10    memory **24<sub>1</sub>**, and performs the above-described actions under the control of the program. The coprocessor **14<sub>1</sub>** may also use the memory **24<sub>1</sub>** as working memory to temporarily store data that the coprocessor generates at intermediate intervals of the second operation.

15    **[9]**                Then, the coprocessors **14<sub>2</sub> – 14<sub>n</sub>** sequentially perform third –  $n^{\text{th}}$  operations on the second –  $(n-1)^{\text{th}}$  results in a manner similar to that discussed above for the coprocessor **14<sub>1</sub>**.

20    **[10]**                The  $n^{\text{th}}$  operation, which is performed by the coprocessor **14<sub>n</sub>**, yields the final result, *i.e.*, the processed data. The coprocessor **14<sub>n</sub>** loads the processed data into a processed-data FIFO (not shown) defined within the memory **26**, and the remote device (not shown in **FIG. 1**) retrieves the processed data from this FIFO.

25    **[11]**                Because the master processor **12** and coprocessors **14** are simultaneously performing different operations of the processing algorithm, the computing machine **10** is often able to process the raw data faster than a computing machine having a single processor that sequentially performs the different operations. Specifically, the single processor cannot retrieve a new set of the raw data until it performs all  $n + 1$  operations on the previous set of raw data. But using the pipeline technique discussed above, the master processor **12** can retrieve a new set of raw data after performing only the first operation. Consequently, for a given clock frequency, this pipeline technique can increase the speed at which the machine **10** processes the raw

data by a factor of approximately  $n + 1$  as compared to a single-processor machine (not shown in **FIG. 1**).

[12] Alternatively, the computing machine **10** may process the raw data in parallel by simultaneously performing  $n + 1$  instances of a processing algorithm, such as an FFT, on the raw data. That is, if the algorithm includes  $n + 1$  sequential operations as described above in the previous example, then each of the master processor **12** and the coprocessors **14** sequentially perform all  $n + 1$  operations on respective sets of the raw data. Consequently, for a given clock frequency, this parallel-processing technique, like the above-described pipeline technique, can increase the speed at which the machine **10** processes the raw data by a factor of approximately  $n + 1$  as compared to a single-processor machine (not shown in **FIG. 1**).

[13] Unfortunately, although the computing machine **10** can process data more quickly than a single-processor computing machine (not shown in **FIG. 1**), the data-processing speed of the machine **10** is often significantly less than the frequency of the processor clock. Specifically, the data-processing speed of the computing machine **10** is limited by the time that the master processor **12** and coprocessors **14** require to process data. For brevity, an example of this speed limitation is discussed in conjunction with the master processor **12**, although it is understood that this discussion also applies to the coprocessors **14**. As discussed above, the master processor **12** executes a program that controls the processor to manipulate data in a desired manner. This program includes a sequence of instructions that the processor **12** executes. Unfortunately, the processor **12** typically requires multiple clock cycles to execute a single instruction, and often must execute multiple instructions to process a single value of data. For example, suppose that the processor **12** is to multiply a first data value A (not shown) by a second data value B (not shown). During a first clock cycle, the processor **12** retrieves a multiply instruction from the memory **22**. During second and third clock cycles, the processor **12** respectively retrieves A and B from the memory **26**. During a fourth clock cycle, the processor **12** multiplies A and B, and, during a fifth clock cycle, stores the resulting product in the memory **22** or **26** or provides the resulting product to the remote device (not shown). This is a best-case scenario, because in

many cases the processor **12** requires additional clock cycles for overhead tasks such as initializing and closing counters. Therefore, at best the processor **12** requires five clock cycles, or an average of 2.5 clock cycles per data value, to process A and B..

**[14]** Consequently, the speed at which the computing machine **10** processes data is often significantly lower than the frequency of the clock that drives the master processor **12** and the coprocessors **14**. For example, if the processor **12** is clocked at 1.0 Gigahertz (GHz) but requires an average of 2.5 clock cycles per data value, then the effective data-processing speed equals  $(1.0 \text{ GHz})/2.5 = 0.4 \text{ GHz}$ . This effective data-processing speed is often characterized in units of operations per second.

Therefore, in this example, for a clock speed of 1.0 GHz, the processor **12** would be rated with a data-processing speed of 0.4 Gigaoperations/second (Gops).

**[15]** **FIG. 2** is a block diagram of a hardwired data pipeline **30** that can typically process data faster than a processor can for a given clock frequency, and often at substantially the same rate at which the pipeline is clocked. The pipeline **30** includes operator circuits **32<sub>1</sub> – 32<sub>n</sub>**, which each perform a respective operation on respective data without executing program instructions. That is, the desired operation is “burned in” to a circuit **32** such that it implements the operation automatically, without the need of program instructions. By eliminating the overhead associated with executing program instructions, the pipeline **30** can typically perform more operations per second than a processor can for a given clock frequency.

**[16]** For example, the pipeline **30** can often solve the following equation faster than a processor can for a given clock frequency:

$$Y(x_k) = (5x_k + 3)2^{x_k}$$

where  $x_k$  represents a sequence of raw data values. In this example, the operator circuit **32<sub>1</sub>** is a multiplier that calculates  $5x_k$ , the circuit **32<sub>2</sub>** is an adder that calculates  $5x_k + 3$ , and the circuit **32<sub>n</sub>** ( $n = 3$ ) is a multiplier that calculates  $(5x_k + 3)2^{x_k}$ .

**[17]** During a first clock cycle  $k=1$ , the circuit **32<sub>1</sub>** receives data value  $x_1$  and multiplies it by 5 to generate  $5x_1$ .

[18] During a second clock cycle  $k = 2$ , the circuit **32<sub>2</sub>** receives  $5x_1$  from the circuit **32<sub>1</sub>** and adds 3 to generate  $5x_1 + 3$ . Also, during the second clock cycle, the circuit **32<sub>1</sub>** generates  $5x_2$ .

[19] During a third clock cycle  $k = 3$ , the circuit **32<sub>3</sub>** receives  $5x_1 + 3$  from the circuit **32<sub>2</sub>** and multiplies by  $2^{x_1}$  (effectively left shifts  $5x_1 + 3$  by  $x_1$ ) to generate the first result  $(5x_1 + 3)2^{x_1}$ . Also during the third clock cycle, the circuit **32<sub>1</sub>** generates  $5x_3$  and the circuit **32<sub>2</sub>** generates  $5x_2 + 3$ .

[20] The pipeline **30** continues processing subsequent raw data values  $x_k$  in this manner until all the raw data values are processed.

10 [21] Consequently, a delay of two clock cycles after receiving a raw data value  $x_1$  — this delay is often called the latency of the pipeline **30** — the pipeline generates the result  $(5x_1 + 3)2^{x_1}$ , and thereafter generates one result — e.g.,  $(5x_2 + 3)2^{x_2}$ ,  $(5x_3 + 3)2^{x_3}$ , . . . ,  $(5x_n + 3)2^{x_n}$  — each clock cycle.

[22] Disregarding the latency, the pipeline **30** thus has a data-processing speed equal to the clock speed. In comparison, assuming that the master processor **12** and coprocessors **14** (**FIG. 1**) have data-processing speeds that are 0.4 times the clock speed as in the above example, the pipeline **30** can process data 2.5 times faster than the computing machine **10** (**FIG. 1**) for a given clock speed.

[23] Still referring to **FIG. 2**, a designer may choose to implement the pipeline **30** in a programmable logic IC (PLIC), such as a field-programmable gate array (FPGA), because a PLIC allows more design and modification flexibility than does an application specific IC (ASIC). To configure the hardwired connections within a PLIC, the designer merely sets interconnection-configuration registers disposed within the PLIC to predetermined binary states. The combination of all these binary states is often called “firmware.” Typically, the designer loads this firmware into a nonvolatile memory (not shown in **FIG. 2**) that is coupled to the PLIC. When one “turns on” the PLIC, it downloads the firmware from the memory into the interconnection-configuration registers. Therefore, to modify the functioning of the PLIC, the designer merely modifies the firmware and allows the PLIC to download the modified firmware into the

interconnection-configuration registers. This ability to modify the PLIC by merely modifying the firmware is particularly useful during the prototyping stage and for upgrading the pipeline **30** “in the field”.

**[24]** Unfortunately, the hardwired pipeline **30** may not be the best choice to execute algorithms that entail significant decision making, particularly nested decision making. A processor can typically execute a nested-decision-making instruction (e.g., a nested conditional instruction such as “if A, then do B, else if C, do D, . . . , else do n”) approximately as fast as it can execute an operational instruction (e.g., “A + B”) of comparable length. But although the pipeline **30** may be able to make a relatively simple decision (e.g., “A > B?”) efficiently, it typically cannot execute a nested decision (e.g., “if A, then do B, else if C, do D, . . . , else do n”) as efficiently as a processor can. One reason for this inefficiency is that the pipeline **30** may have little on-board memory, and thus may need to access external working/program memory (not shown). And although one may be able to design the pipeline **30** to execute such a nested decision, the size and complexity of the required circuitry often makes such a design impractical, particularly where an algorithm includes multiple different nested decisions.

**[25]** Consequently, processors are typically used in applications that require significant decision making, and hardwired pipelines are typically limited to “number crunching” applications that entail little or no decision making.

**[26]** Furthermore, as discussed below, it is typically much easier for one to design/modify a processor-based computing machine, such as the computing machine **10** of **FIG. 1**, than it is to design/modify a hardwired pipeline such as the pipeline **30** of **FIG. 2**, particularly where the pipeline **30** includes multiple PLICs.

**[27]** Computing components, such as processors and their peripherals (e.g., memory), typically include industry-standard communication interfaces that facilitate the interconnection of the components to form a processor-based computing machine.

**[28]** Typically, a standard communication interface includes two layers: a physical layer and a services layer.

**[29]** The physical layer includes the circuitry and the corresponding circuit interconnections that form the interface and the operating parameters of this circuitry. For example, the physical layer includes the pins that connect the component to a bus, the buffers that latch data received from the pins, and the drivers that drive signals onto the pins. The operating parameters include the acceptable voltage range of the data signals that the pins receive, the signal timing for writing and reading data, and the supported modes of operation (e.g., burst mode, page mode). Conventional physical layers include transistor-transistor logic (TTL) and RAMBUS.

**[30]** The services layer includes the protocol by which a computing component transfers data. The protocol defines the format of the data and the manner in which the component sends and receives the formatted data. Conventional communication protocols include file-transfer protocol (FTP) and transmission control protocol/internet protocol (TCP/IP).

**[31]** Consequently, because manufacturers and others typically design computing components having industry-standard communication interfaces, one can typically design the interface of such a component and interconnect it to other computing components with relatively little effort. This allows one to devote most of his time to designing the other portions of the computing machine, and to easily modify the machine by adding or removing components.

**[32]** Designing a computing component that supports an industry-standard communication interface allows one to save design time by using an existing physical-layer design from a design library. This also insures that he/she can easily interface the component to off-the-shelf computing components.

**[33]** And designing a computing machine using computing components that support a common industry-standard communication interface allows the designer to interconnect the components with little time and effort. Because the components support a common interface, the designer can interconnect them via a system bus with little design effort. And because the supported interface is an industry standard, one can easily modify the machine. For example, one can add different components and

peripherals to the machine as the system design evolves, or can easily add/design next-generation components as the technology evolves. Furthermore, because the components support a common industry-standard services layer, one can incorporate into the computing machine's software an existing software module that implements the corresponding protocol. Therefore, one can interface the components with little effort because the interface design is essentially already in place, and thus can focus on designing the portions (e.g., software) of the machine that cause the machine to perform the desired function(s).

**[34]** But unfortunately, there are no known industry-standard services layers for components, such as PLICs, used to form hardwired pipelines such as the pipeline **30** of **FIG. 2**.

**[35]** Consequently, to design a pipeline having multiple PLICs, one typically spends a significant amount of time and exerts a significant effort designing "from scratch" and debugging the services layer of the communication interface between the PLICs. Typically, such an ad hoc services layer depends on the parameters of the data being transferred between the PLICs. Likewise, to design a pipeline that interfaces to a processor, one would have to spend a significant amount of time and exert a significant effort in designing and debugging the services layer of the communication interface between the pipeline and the processor.

**[36]** Similarly, to modify such a pipeline by adding a PLIC to it, one typically spends a significant amount of time and exerts a significant effort designing and debugging the services layer of the communication interface between the added PLIC and the existing PLICs. Likewise, to modify a pipeline by adding a processor, or to modify a computing machine by adding a pipeline, one would have to spend a significant amount of time and exert a significant effort in designing and debugging the services layer of the communication interface between the pipeline and processor.

**[37]** Consequently, referring to **FIGS. 1** and **2**, because of the difficulties in interfacing multiple PLICs and in interfacing a processor to a pipeline, one is often forced to make significant tradeoffs when designing a computing machine. For

example, with a processor-based computing machine, one is forced to trade number-crunching speed and design/modification flexibility for complex decision-making ability. Conversely, with a hardwired pipeline-based computing machine, one is forced to trade complex-decision-making ability and design/modification flexibility for number-crunching speed. Furthermore, because of the difficulties in interfacing multiple PLICs, it is often impractical for one to design a pipeline-based machine having more than a few PLICs. As a result, a practical pipeline-based machine often has limited functionality. And because of the difficulties in interfacing a processor to a PLIC, it would be impractical to interface a processor to more than one PLIC. As a result, the benefits obtained by combining a processor and a pipeline would be minimal.

[38] Therefore, a need has arisen for a new computing architecture that allows one to combine the decision-making ability of a processor-based machine with the number-crunching speed of a hardwired-pipeline-based machine.

#### SUMMARY

[39] According to an embodiment of the invention, a pipeline accelerator includes a bus and a plurality of pipeline units each coupled to the bus and each including at least one respective hardwired-pipeline circuit.

[40] By including a plurality of pipeline units in the pipeline accelerator, one can increase the accelerator's data-processing performance as compared to a single-pipeline-unit accelerator. Furthermore, by designing the pipeline units so that they communicate with each other and with other peers via a common bus, one can alter the number of pipeline units, and thus alter the configuration and functionality of the accelerator, by merely coupling or uncoupling pipeline units to or from the bus. This eliminates the need to design or redesign the pipeline-unit interfaces each time one alters one of the pipeline units or alters the number of pipeline units within the accelerator.

#### BRIEF DESCRIPTION OF THE DRAWINGS

[41] FIG. 1 is a block diagram of a computing machine having a conventional multi-processor architecture.

[42] FIG. 2 is a block diagram of a conventional hardwired pipeline.

[43] FIG. 3 is a block diagram of a computing machine having a peer-vector architecture according to an embodiment of the invention.

[44] FIG. 4 is a block diagram of a pipeline unit of the pipeline accelerator of  
5 FIG. 3 according to an embodiment of the invention.

[45] FIG. 5 is a block diagram of a pipeline unit of the pipeline accelerator of FIG. 3 according to another embodiment of the invention.

[46] FIG. 6 is a block diagram of the pipeline accelerator of FIG. 3 including multiple pipeline units according to an embodiment of the invention.

10 [47] FIG. 7 is a block diagram of the pipeline accelerator of FIG. 3 including multiple pipeline units according to another embodiment of the invention.

[48] FIG. 8 is a block diagram of the pipeline accelerator of FIG. 3 including groups of multiple pipeline units according to an embodiment of the invention.

#### DETAILED DESCRIPTION

15 [49] FIG. 3 is a schematic block diagram of a computing machine **40**, which has a peer-vector architecture according to an embodiment of the invention. In addition to a host processor **42**, the peer-vector machine **40** includes a pipeline accelerator **44**, which performs at least a portion of the data processing, and which thus effectively replaces the bank of coprocessors **14** in the computing machine **10** of FIG. 1.

20 Therefore, the host-processor **42** and the accelerator **44** (or units thereof as discussed below) are “peers” that can transfer data vectors back and forth. Because the accelerator **44** does not execute program instructions, it typically performs mathematically intensive operations on data significantly faster than a bank of coprocessors can for a given clock frequency. Consequently, by combining the  
25 decision-making ability of the processor **42** and the number-crunching ability of the accelerator **44**, the machine **40** has the same abilities as, but can often process data faster than, a conventional computing machine such as the machine **10**. Furthermore, as discussed below, providing the accelerator **44** with a communication interface that is

compatible with the communication interface of the host processor **42** facilitates the design and modification of the machine **40**, particularly where the processor's communication interface is an industry standard. And where the accelerator **44** includes multiple pipeline units (e.g., PLIC-based circuits), providing each of these units with the same communication interface facilitates the design and modification of the accelerator, particularly where the communication interfaces are compatible with an industry-standard interface. Moreover, the machine **40** may also provide other advantages as described below and in the previously cited patent applications.

[50] Still referring to **FIG. 3**, in addition to the host processor **42** and the pipeline accelerator **44**, the peer-vector computing machine **40** includes a processor memory **46**, an interface memory **48**, a pipeline bus **50**, one or more firmware memories **52**, an optional raw-data input port **54**, a processed-data output port **58**, and an optional router **61**.

[51] The host processor **42** includes a processing unit **62** and a message handler **64**, and the processor memory **46** includes a processing-unit memory **66** and a handler memory **68**, which respectively serve as both program and working memories for the processor unit and the message handler. The processor memory **46** also includes an accelerator-configuration registry **70** and a message-configuration registry **72**, which store respective configuration data that allow the host processor **42** to configure the functioning of the accelerator **44** and the format of the messages that the message handler **64** sends and receives.

[52] The pipeline accelerator **44** is disposed on at least one PLIC (**FIG. 4**) and includes hardwired pipelines **74<sub>1</sub> – 74<sub>n</sub>**, which process respective data without executing program instructions. The firmware memory **52** stores the configuration firmware for the accelerator **44**. If the accelerator **44** is disposed on multiple PLICs, then these PLICs and their respective firmware memories may be disposed in multiple pipeline units, which are discussed further below in conjunction with **FIGS. 4 – 8**. Alternatively, the accelerator **44** may be disposed on at least one ASIC, and thus may have internal interconnections that are unconfigurable once the ASIC is formed. In this alternative,

the machine **40** may omit the firmware memory **52**. Furthermore, although the accelerator **44** is shown including multiple pipelines **74<sub>1</sub>-74<sub>n</sub>**, it may include only a single pipeline. In addition, although not shown, the accelerator **44** may include one or more processors such as a digital-signal processor (DSP). Moreover, although not shown,  
5 the accelerator **44** may include a data input port and/or a data output port.

**[53]** And although the host processor **42** and pipeline accelerator **44** are discussed as being disposed on different ICs, the host processor and pipeline accelerator may be disposed on the same IC.

**[54]** The general operation of the peer-vector machine **40** is discussed in  
10 previously cited U.S. Publication No. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, the structure and operation of the host processor **42** is discussed in previously cited U.S. Publication No. 2004/0181621 entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, and the structure and  
15 operation of the pipeline accelerator **44** is discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD (Attorney Docket No. 1934-13-3) and below in conjunction with **FIGS. 4 – 8**.

**[55]** **FIG. 4** is a block diagram of a unit **78** of the pipeline accelerator **44** of  
20 **FIG. 3** according to an embodiment of the invention.

**[56]** The accelerator **44** includes one or more such pipeline units **78** (only one shown in **FIG. 4**), each of which includes a pipeline circuit **80**, such as a PLIC or an ASIC. As discussed further below and in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING  
25 ARCHITECTURE AND RELATED SYSTEM AND METHOD, each pipeline unit **78** is a “peer” of the host processor **42** (**FIG. 3**) and of the other pipeline units of the accelerator **44**. That is, each pipeline unit **78** can communicate directly with the host processor **42** or with any other pipeline unit. Thus, this peer-vector architecture prevents data “bottlenecks” that otherwise might occur if all of the pipeline units **78** communicated

through a central location such as a master pipeline unit (not shown) or the host processor **42**. Furthermore, this architecture allows one to add or remove peers from the peer-vector machine **40** (**FIG. 3**) without significant modifications to the machine.

**[57]** The pipeline circuit **80** includes a communication interface **82**, which  
5 transfers data between a peer, such as the host processor **42** (**FIG. 3**), and the following other components of the pipeline circuit: the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>** via a communication shell **84**, a pipeline controller **86**, an exception manager **88**, and a configuration manager **90**. The pipeline circuit **80** may also include an  
10 industry-standard bus interface **91** and a communication bus **93**, which connects the interface **82** to the interface **91**. Alternatively, the functionality of the interface **91** may be included within the communication interface **82** and the bus **93** omitted.

**[58]** By designing the components of the pipeline circuit **80** as separate modules, one can often simplify the design of the pipeline circuit. That is, one can design and test each of these components separately, and then integrate them much  
15 like one does when designing software or a processor-based computing system (such as the system **10** of **FIG. 1**). In addition, one can save in a library (not shown) hardware description language (HDL) that defines these components — particularly components, such as the communication interface **82**, that designers will probably use frequently in other pipeline designs — thus reducing the design and test time of future pipeline  
20 designs that use the same components. That is, by using the HDL from the library, the designer need not redesign previously implemented components from scratch, and thus can focus his efforts on the design of components that were not previously implemented, or on the modification of previously implemented components. Moreover, one can save in the library HDL that defines multiple versions of the pipeline circuit **80**  
25 or multiple versions of the entire pipeline accelerator **44** so that one can pick and choose among existing designs.

**[59]** Still referring to **FIG. 4**, the communication interface **82** sends and receives (via the bus interface **91** of present) data in a format recognized by the message handler **64** (**FIG. 3**), and thus typically facilitates the design and modification

of the peer-vector machine **40** (**FIG. 3**). For example, if the data format is an industry standard such as the Rapid I/O format, then one need not design a custom interface between the host processor **42** and the pipeline unit **78**. Furthermore, by allowing the pipeline unit **78** to communicate with other peers, such as the host processor **42** (**FIG. 3**), via the pipeline bus **50** instead of via a non-bus interface, one can change the number of pipeline units by merely connecting or disconnecting them (or the circuit cards that hold them) to the pipeline bus instead of redesigning a non-bus interface from scratch each time a pipeline unit is added or removed.

[60] The hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>** perform respective operations on data as discussed above in conjunction with **FIG. 3** and in previously cited U.S. Publication Nos. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD and 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, and the communication shell **84** interfaces the pipelines to the other components of the pipeline circuit **80** and to other circuits (such as a data memory **92** discussed below) of the pipeline unit **78**.

[61] The controller **86** synchronizes the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>** in response to SYNC signals and special pipeline-bus communications (*i.e.*, “events”) from other peers, and monitors and controls the sequence in which the pipelines perform their respective data operations. For example, a peer, such as the host processor **42**, may pulse a SYNC signal or send an event to the pipeline unit **78** via the pipeline bus **50** to indicate that the peer has finished sending a block of data to the pipeline unit and to cause the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>** to begin processing this data. Typically, a SYNC signal is used to synchronize a time-critical operation, and an event is used to synchronize a non-time-critical operation. Typically, an event is a data-less communication that is often called a “doorbell”. But an event may include data, in which case it is often called an “event message”. SYNC signals and events are discussed further in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

**[62]** The exception manager **88** monitors the status of the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>**, the communication interface **82**, the communication shell **84**, the controller **86**, and the bus interface **91** (if present), and reports exceptions to the host processor **42** (**FIG. 3**). For example, if a buffer in the communication interface **82** overflows, then the exception manager **88** reports this to the host processor **42**. The exception manager may also correct, or attempt to correct, the problem giving rise to the exception. For example, for an overflowing buffer, the exception manager **88** may increase the size of the buffer, either directly or via the configuration manager **90** as discussed below.

**[63]** The configuration manager **90** sets the soft configuration of the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>**, the communication interface **82**, the communication shell **84**, the controller **86**, the exception manager **88**, and the interface **91** (if present) in response to soft-configuration data from the host processor **42** (**FIG. 3**) — as discussed in previously cited U.S. Publication No. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, the hard configuration denotes the actual topology, on the transistor and circuit-block level, of the pipeline circuit **80**, and the soft configuration denotes the physical parameters (e.g., data width, table size) of the hard-configured components. That is, soft configuration data is similar to the data that can be loaded into a register of a processor (not shown in **FIG. 4**) to set the operating mode (e.g., burst-memory mode) of the processor. For example, the host processor **42** may send soft-configuration data that causes the configuration manager **90** to set the number and respective priority levels of data and event queues in the communication interface **82**. The exception manager **88** may also send soft-configuration data that causes the configuration manager **90** to, e.g., increase the size of an overflowing buffer in the communication interface **82**.

**[64]** The industry-standard bus interface **91** is a conventional bus-interface circuit that reduces the size and complexity of the communication interface **82** by effectively offloading some of the interface circuitry from the communication interface. Therefore, if one wishes to change the parameters of the pipeline bus **50** or router **61** (**FIG. 3**), then he need only modify the interface **91** and not the communication interface

**82.** Alternatively, one may dispose the interface **91** in an IC (not shown) that is external to the pipeline circuit **80**. Offloading the interface **91** from the pipeline circuit **80** frees up resources on the pipeline circuit for use in, e.g., the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>** and the controller **86**. Or, as discussed above, the bus interface **91** may be part of the communication interface **82**.

**[65]** Still referring to **FIG. 4**, in addition to the pipeline circuit **80**, the pipeline unit **78** of the accelerator **44** includes the data memory **92**, and, if the pipeline circuit is a PLIC, a firmware memory **52**.

**[66]** The data memory **92** buffers data as it flows between another peer, such as the host processor **42** (**FIG. 3**), and the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>**, and is also a working memory for the hardwired pipelines. The communication interface **82** interfaces the data memory **92** to the pipeline bus **50** (via the communication bus **94** and industry-standard interface **91** if present), and the communication shell **84** interfaces the data memory to the hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>**.

**[67]** The data memory **92** (or other portion of the pipeline unit **78**) may also store a profile of the pipeline unit. The profile describes the pipeline unit **78** sufficiently for the host processor **42** (**FIG. 3**) to appropriately configure itself, the pipeline unit, and other peers of the peer-vector machine **40** (**FIG. 3**) for intercommunication. For example, the profile may identify the data operations and communication protocols that the pipeline unit **78** is capable of implementing. Consequently, by reading the profile during initialization of the peer-vector machine **40**, the host processor **42** can properly configure the message handler **64** (**FIG. 3**) to communicate with the pipeline unit **78**. This technique is analogous to the “plug and play” technique by which a computer can configure itself to communicate with a newly installed peripheral such as a disk drive. Configuration of the host processor **42** and pipeline unit **78** are discussed further in previously cited U.S. Publication Nos. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD and 2004/0170070 entitled PROGRAMMABLE CIRCUIT AND RELATED COMPUTING MACHINE AND METHOD.

[68] As discussed above in conjunction with **FIG. 3**, where the pipeline circuit **80** is a PLIC, the firmware memory **52** stores the firmware that sets the hard configuration of the pipeline circuit. The memory **52** loads the firmware into the pipeline circuit **80** during the configuration of the accelerator **44**, and may receive modified  
5 firmware from the host processor **42** (**FIG. 3**) via the communication interface **82** during or after the configuration of the accelerator. The loading and receiving of firmware is further discussed in previously cited U.S. Publication Nos. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, 2004/0181621 entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING  
10 ARCHITECTURE AND RELATED SYSTEM AND METHOD, and 2004/0170070 entitled PROGRAMMABLE CIRCUIT AND RELATED COMPUTING MACHINE AND METHOD.

[69] Still referring to **FIG. 4**, the pipeline unit **78** may include a circuit board or card **98** on which are disposed the pipeline circuit **80**, data memory **92**, and firmware memory **52**. The circuit board **98** may be plugged into a pipeline-bus connector (not  
15 shown) much like a daughter card can be plugged into a slot of a mother board in a personal computer (not shown). Although not shown, the pipeline unit **78** may include conventional ICs and components such as a power regulator and a power sequencer; these ICs/components may also be disposed on the card **98** as is known.

[70] Further details of the structure and operation of the pipeline unit **78** are  
20 discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[71] **FIG. 5** is a block diagram of a pipeline unit **100** of the pipeline accelerator **44** of **FIG. 3** according to another embodiment of the invention. The pipeline unit **100** is  
25 similar to the pipeline unit **78** of **FIG. 4** except that the pipeline unit **100** includes multiple pipeline circuits **80** — here two pipeline circuits **80a** and **80b**. Increasing the number of pipeline circuits **80** typically allows an increase in the number  $n$  of hardwired pipelines **74<sub>1</sub>-74<sub>n</sub>**, and thus an increase in the functionality of the pipeline unit **100** as compared to the pipeline unit **78**. Furthermore, the pipeline unit **100** includes a firmware

memory **52a** for the pipeline circuit **80a** and a firmware memory **52b** for the pipeline circuit **80b**. Alternatively, the pipeline circuits **80a** and **80b** may share a single firmware memory.

[72] In the pipeline unit **100**, the services components, *i.e.*, the communication interface **82**, the controller **86**, the exception manager **88**, the configuration manager **90**, and the optional industry-standard bus interface **91**, are disposed on the pipeline circuit **80a**, and the pipelines **74<sub>1</sub>-74<sub>n</sub>** and the communication shell **84** are disposed on the pipeline circuit **80b**. By locating the services components and the pipelines **74<sub>1</sub>-74<sub>n</sub>** on separate pipeline circuits **80a** and **80b**, respectively, one can include a higher number *n* of pipelines and/or more complex pipelines than he/she can where the service components and the pipelines are located on the same pipeline circuit. Alternatively, the portion of the communication shell **84** that interfaces the pipelines **74<sub>1</sub>-74<sub>n</sub>** to the interface **82** and to the controller **86** may be disposed on the pipeline circuit **80a**.

[73] Further details of the structure and operation of the pipeline unit **100** are discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[74] FIG. 6 is a block diagram of the accelerator **44** of FIG. 3 having multiple pipeline units **78** (FIG. 4) or **100** (FIG. 5) according to an embodiment of the invention. For clarity, the accelerator **44** is discussed as having multiple pipeline units **78<sub>1</sub> – 78<sub>n</sub>**, it being understood that the accelerator may include multiple pipeline units **100** or a combination of units **78** and **100**. By including multiple pipeline units **78**, one can increase the functionality and processing power of the accelerator **44** as compared to an accelerator having only one pipeline unit. Furthermore, because each pipeline unit **78** typically has a common industry-standard interface, one can easily modify the accelerator **44** by adding or removing pipeline units.

[75] In one implementation of the multiple-pipeline accelerator **44**, the industry-standard bus interface **91** is omitted from each pipeline unit **78<sub>1</sub> – 78<sub>n</sub>**, and a single external (to the pipeline units) interface **91** and communication bus **94** are

common to all of the pipeline units. Including a single external bus interface **91** frees up resources on the pipeline circuits **80** (**FIG. 4**) as discussed above in conjunction with **FIG. 4**. The pipeline units **78<sub>1</sub> – 78<sub>n</sub>** may all be disposed on a single circuit board (not shown in **FIG. 6**), each pipeline unit may be disposed on a respective circuit board, or  
5 groups of multiple pipeline units may be respectively disposed on multiple circuit boards. In the latter two implementations, the bus interface **91** is disposed on one of the circuit boards. Alternatively, the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** may each include a respective industry-standard bus interface **91** as discussed above in conjunction with **FIG. 4**, and thus may each communicate directly with the pipeline bus **50** or router **61**  
10 (**FIG. 3**). In this implementation, the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** may be disposed on a single or multiple circuit boards as discussed above.

[76] Each of the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** is a peer of the host processor **42** (**FIG. 3**) and of each other. That is, each pipeline unit **78** can communicate directly with any other pipeline unit via the communication bus **94**, and can communicate with the  
15 host processor **42** via the communication bus **94**, the bus interface **91**, the router **61** (if present), and the pipeline bus **50**. Alternatively, where the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** each include a respective bus interface **91**, then each pipeline unit can communicate directly with the host processor **42** via the router **61** (if present) and the pipeline bus **50**.

[77] The operation of the multi-pipeline-unit accelerator **44** is now described by  
20 way of two examples.

[78] In a first example, the pipeline unit **78<sub>1</sub>** transfers data to the pipeline **78<sub>n</sub>**, which processes the data in a time-critical manner; thus, the pipeline units **78<sub>1</sub>** and **78<sub>n</sub>** use one or more SYNC signals to synchronize the data transfer and processing. Typically, a SYNC signal is fast enough to trigger a time-critical function, but requires  
25 significant hardware resources; comparatively, an event typically is not fast enough to trigger a time-critical function, but requires significantly fewer hardware resources. As discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, because a SYNC signal is routed directly from peer to peer, it

can trigger a function more quickly than an event, which traverses, e.g., the pipeline bus **50** (**FIG. 3**), and the communication bus **94**. But because they are separately routed, the SYNC signals require dedicated circuitry, such as routing lines and buffers of the pipeline circuit **80** (**FIG. 4**). Conversely, because they use the existing data-transfer infrastructure (e.g. the pipeline bus **50** and the communication bus **94**), the events require fewer dedicated hardware resources. Consequently, designers tend to use events to trigger all but the most time-critical functions.

[79] First, the pipeline unit **78<sub>1</sub>** sends data to the pipeline unit **78<sub>n</sub>** by driving the data onto the communication bus **94**. Typically, the pipeline unit **78<sub>1</sub>** generates a message that includes the data and a header that contains the address of the pipeline unit **78<sub>n</sub>**. If the pipeline unit **78<sub>1</sub>** were to send the data to multiple pipeline units **78**, then it may do so in one of two ways. Specifically, the pipeline unit **78<sub>1</sub>** may sequentially send separate messages to each of the destination pipeline units **78**, each message including a header containing the address of a respective destination unit. Alternatively, the pipeline unit **78<sub>1</sub>** may simultaneously send the data to each of the destination pipeline units **78** by including in single message the data and a header containing the addresses of each destination pipeline unit. The sending of data is further discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[80] Next, the pipeline unit **78<sub>n</sub>** receives the data. Because the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** are each coupled to the common communication bus **94**, each pipeline unit **78<sub>2</sub> – 78<sub>n</sub>** determines whether or not it is an intended recipient of the data. For example, each pipeline unit **78<sub>2</sub> – 78<sub>n</sub>** determines whether its address is included in the header of the message. In this example, the units **78<sub>2</sub> – 78<sub>n-1</sub>** determine that they are not intended recipients of the data, and thus ignore the data, i.e., do not load the data into their data memories **92** (**FIG. 4**). Conversely, the pipeline unit **78<sub>n</sub>** determines that it is an intended recipient of the data, and thus loads the data into its data memory **92**. The receiving of data is discussed further in U.S. Publication No. 2004/0136241 entitled

## PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[81] Then, when the pipeline unit  $78_n$  is ready to process the received data, a peer, such as the pipeline unit  $78_1$ , or an external device (not shown) pulses a SYNC signal to cause the pipeline unit  $78_n$  to process the data in a timely manner. There exist a number of techniques by which the peer/device that pulses the SYNC signal may determine when the pipeline unit  $78_n$  is ready to process the received data. For example, the peer/device may merely pulse the SYNC signal a predetermined time after the pipeline unit  $78_1$  sends the data. Presumably, the predetermined time is long enough to allow the pipeline unit  $78_n$  to receive and load the data into its data memory 92 (FIG. 4). Alternatively, the pipeline unit  $78_n$  may pulse a SYNC signal to inform the peer/device that it is ready to process the received data.

[82] Next, in response to the pulsed SYNC signal, the pipeline unit  $78_n$  processes the received data. The processing of data by a pipeline unit is discussed further in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[83] Then, when the pipeline unit  $78_n$  is finished processing the data, a peer, an external device (not shown), or the unit  $78_n$  itself may pulse a SYNC signal to notify the pipeline unit  $78_1$  to send more data.

[84] In a second example, the host processor 42 (FIG. 3) transfers data to the pipeline  $78_n$ , which processes the data in a non-time-critical manner; thus the host processor and the pipeline unit  $78_n$  use one or more events to synchronize the data transfer and processing for the reasons discussed above.

25 [85] First, the host processor 42 (FIG. 3) sends data to the pipeline unit  $78_n$  by driving the data onto the pipeline bus 50 (FIG. 3). Typically, the host processor 42 generates a message that includes the data and a header containing the address of the pipeline unit  $78_n$ . If the host processor 42 were to send the data to multiple pipeline

units **78**, then it may do so in one of the two ways discussed above in conjunction with the first example.

[86]           Next, the pipeline unit **78<sub>n</sub>** receives the data from the pipeline bus **50** (FIG. 3) via the industry-standard bus interface **91** and the communication bus **94**. Because the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** are each coupled to the common communication bus **94**, each pipeline unit determines whether it is an intended recipient of the data in the manner discussed above in conjunction with the first example.

[87]           Then, when the pipeline unit **78<sub>n</sub>** is ready to process the received data, a peer, such as the host processor **42** (FIG. 3), or an external device (not shown), generates an event on the pipeline bus **50** or directly on the communication bus **94** to cause the pipeline unit **78<sub>n</sub>** to process the data in a timely manner. There exist a number of techniques by which the peer/device that generates the event may determine when the pipeline unit **78<sub>n</sub>** is ready to process the received data. For example, the peer/device may merely generate the event a predetermined time after the host processor **42** sends the data. Presumably, the predetermined time is long enough to allow the pipeline unit **78<sub>n</sub>** to receive and load the data into its data memory **92** (FIG. 4). Alternatively, the pipeline unit **78<sub>n</sub>** may generate an event to inform the peer/device that it is ready to process the received data.

[88]           Next, the pipeline unit **78<sub>n</sub>** receives the event. The receiving of an event is discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[89]           Then, in response to the received event, the pipeline unit **78<sub>n</sub>** processes the received data. The processing of data by a pipeline unit **78<sub>n</sub>** is discussed further in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[90] Next, when the pipeline unit  $78_n$  is finished processing the data, a peer, an external device (not shown), or the unit  $78_n$  itself may generate an event to notify the host processor **42** (**FIG. 3**) to send more data.

[91] Still referring to **FIG. 6**, alternative implementations of the accelerator **44** are contemplated. For example, although the above-discussed first and second examples of operation respectively use SYNC signals and events exclusively, it is contemplated that the accelerator **44** can use both SYNC signals and events in combination. Furthermore, other peers can use one or more of the multiple pipeline units **78** or **100** merely for bulk storage of data in their respective data memories **92**. In addition, a designer may replace the host processor **42** (**FIG. 3**) with one or more of the pipeline units **78** or **100**, which together form a “host” peer that performs the functions of the host processor. Moreover, one or more of the pipeline units **78** or **100** may act as one or more message-distribution peers. For example, suppose that the host processor **42** generates a message for transmission to multiple subscriber peers. The host processor **42** may send the message to a message-distribution peer, which then distributes the message to each of the subscriber peers. Consequently, the message-distribution peer, not the host processor **42**, undertakes the burden of distributing the message, and thus allows the host processor to devote more time and resources to other tasks.

[92] **FIG. 7** is a block diagram of the accelerator **44** (**FIG. 3**) having multiple pipeline units **78** (**FIG. 4**) or **100** (**FIG. 5**) according to another embodiment of the invention.

[93] The accelerator **44** of **FIG. 7** is the same as the accelerator **44** of **FIG. 6**, except that the accelerator **44** of **FIG. 7** includes a communication-bus router **110** for routing data between the pipeline units  $78_1 - 78_n$  and other peers, such as the host processor **42** (**FIG. 3**), and devices (not shown) that are coupled to the pipeline bus **50** (**FIG. 3**). For clarity, the accelerator **44** of **FIG. 7** is discussed as having multiple pipeline units  $78_1 - 78_n$ , it being understood that the accelerator may include multiple pipeline units **100** or a combination of units **78** and **100**.

[94] The communication-bus router **110** is coupled to the pipeline units **78<sub>1</sub> – 78<sub>n</sub>** via respective branches **94<sub>1</sub> – 94<sub>n</sub>** of the communication bus **94**, and is coupled to the industry-standard bus interface **91** (if present) via a bus **112**. Alternatively, as discussed above in conjunction with **FIG. 6**, each pipeline unit **78<sub>1</sub> – 78<sub>n</sub>** may include a  
 5 respective interface **91** on board, and thus the external interface **91** can be omitted such that the router **110** is directly coupled to the pipeline bus **50** (or router **61** if present) of **FIG. 3**.

[95] The router **110** routes signals from the pipeline bus **50** (**FIG. 3**) to the respective destination pipeline unit or units **78<sub>1</sub> – 78<sub>n</sub>**, and also routes signals from a  
 10 source pipeline unit to one or more destination pipeline units or to the pipeline bus. Consequently, the router **110** offloads from each of the pipeline units **78<sub>1</sub>–78<sub>n</sub>** the function of determining whether a signal on the communication bus **94** is intended for that pipeline unit. This offloading may free up resources on the pipeline circuit **80** (**FIG. 4**) of each pipeline unit **78<sub>1</sub> – 78<sub>n</sub>**, and thus may allow an increase in the functionality of  
 15 each pipeline unit.

[96] Still referring to **FIG. 7**, the operation of the multi-pipeline-unit accelerator **44** with the router **110** is now described. Because the operation is similar to that described above for the accelerator **44** of **FIG. 6**, the below description highlights the operational differences between the accelerators of **FIGS. 6** and **7**.

20 [97] In a first example, the pipeline unit **78<sub>1</sub>** transfers data to the pipeline **78<sub>n</sub>**, which processes the data in a time-critical manner; thus, the pipeline units **78<sub>1</sub>** and **78<sub>n</sub>** use one or more SYNC signals to synchronize the data transfer and processing as discussed above in conjunction with the first example of **FIG. 6**.

[98] First, the pipeline unit **78<sub>1</sub>** sends data to the pipeline unit **78<sub>n</sub>** by driving the  
 25 data onto the branch **94<sub>1</sub>** of the communication bus. Typically, the pipeline unit **78<sub>1</sub>** generates a message that includes the data and a header that contains the address of the pipeline unit **78<sub>n</sub>**.

[99] Next, the router **110** receives the data, determines that the destination of the data is the pipeline unit **78<sub>n</sub>**, and drives the data onto the branch **94<sub>n</sub>** of the

communication bus. Typically, the router **110** determines the destination of the data by analyzing the header of the message containing the data and extracting the destination address from the header. Consequently, because the router **110** determines the proper destination of the data, the pipeline unit **78<sub>n</sub>** can merely accept the data from the router  
5 without determining whether it is an intended recipient of the data. Alternatively, the pipeline **78<sub>n</sub>** may determine whether it is an intended recipient of the data, and generate an exception (discussed in previously cited U.S. Publication Nos. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, 2004/0181621 entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING  
10 ARCHITECTURE AND RELATED SYSTEM AND METHOD, and 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD if it is not an intended recipient. The pipeline unit **78<sub>n</sub>** can send this exception to the host processor **42 (FIG. 3)** via the router **110**, the industry-standard bus interface **91** (if present), the router **61** (if present), and the  
15 pipeline bus **50 (FIG. 3)**.

**[100]** Then, the pipeline unit **78<sub>n</sub>** loads the data from the bus branch **94<sub>n</sub>**. The loading of data by a pipeline unit is further discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

20 **[101]** Next, when the pipeline unit **78<sub>n</sub>** is ready to process the received data, a peer, such as the pipeline unit **78<sub>1</sub>**, or an external device (not shown) pulses a SYNC signal to cause the pipeline unit **78<sub>n</sub>** to process the data in a timely manner as discussed above in conjunction with the first example of **FIG. 6**.

**[102]** Then, in response to the pulsed SYNC signal, the pipeline unit **78<sub>n</sub>**  
25 processes the received data as discussed above in conjunction with the first example of **FIG. 6**.

**[103]** Next, when the pipeline unit **78<sub>n</sub>** is finished processing the data a peer, an external device (not shown), or the unit **78<sub>n</sub>** itself may pulse a SYNC signal to notify the pipeline unit **78<sub>1</sub>** to send more data.

[104] In a second example, the host processor **42 (FIG. 3)** transfers data to the pipeline **78<sub>n</sub>**, which processes the data in a non-time-critical manner; thus the host processor and the pipeline unit **78<sub>n</sub>** use one or more events to synchronize the data transfer and processing for the reasons discussed above in conjunction with **FIG. 6**.

5 [105] First, the host processor **42 (FIG. 3)** sends data to the pipeline unit **78<sub>n</sub>** by driving the data onto the pipeline bus **50 (FIG. 3)**. Typically, the host processor **42** generates a message that includes the data and a header containing the address of the pipeline unit **78<sub>n</sub>**.

10 [106] Next, the router **110** receives the data from the pipeline bus **50 (FIG. 3)** via the industry-standard bus interface **91** (if present) and the bus **112**.

[107] Then, the router **110** determines that the destination of the data is the pipeline unit **78<sub>n</sub>**, and drives the data onto the branch **94<sub>n</sub>** of the communication bus. Typically, the router **110** determines the destination of the header as discussed above in conjunction with the first example of **FIG. 7**. Consequently, because the router **110** determines the proper destination of the data, the pipeline unit **78<sub>n</sub>** can merely accept the data from the router without determining whether it is an intended recipient of the data. Alternatively, the pipeline **78<sub>n</sub>** may determine whether it is an intended recipient of the data, generate an exception (discussed in previously cited U.S. Publication Nos. 2004/0133763 entitled IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, 2004/0181621 entitled COMPUTING MACHINE HAVING IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD, and 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD if it is not an intended recipient, and send the exception to the host processor **42 (FIG. 3)** as discussed above in conjunction with the second example of **FIG. 6**.

[108] Next, the pipeline unit **78<sub>n</sub>** loads the data from the bus branch **94<sub>n</sub>**. The loading of data by a pipeline unit is further discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[109] Then, when the pipeline unit **78<sub>n</sub>** is ready to process the received data, a peer, such as the host processor **42** (**FIG. 3**), or an external device (not shown), generates an event on the pipeline bus **50** or on one of the branches **94<sub>1</sub> – 94<sub>n-1</sub>** of the communication bus to cause the unit **78<sub>n</sub>** to process the data in a timely manner as discussed above in conjunction with the second example of **FIG. 6**.

[110] Next, the router **110** receives the event, determines that it is intended for the pipeline unit **78<sub>n</sub>**, and drives the event onto the bus branch **94<sub>n</sub>**.

[111] Then, the pipeline unit **78<sub>n</sub>** loads the event from the bus branch **94**. The loading of an event by a pipeline unit **78** is discussed in previously cited U.S. Publication No. 2004/0136241 entitled PIPELINE ACCELERATOR FOR IMPROVED COMPUTING ARCHITECTURE AND RELATED SYSTEM AND METHOD.

[112] Next, in response to the received event, the pipeline unit **78<sub>n</sub>** processes the received data.

[113] Then, when the pipeline unit **78<sub>n</sub>** is finished processing the data, a peer, an external device (not shown), or the unit **78<sub>n</sub>** itself may generate an event to notify the host processor **42** (**FIG. 3**) to send more data.

[114] Still referring to **FIG. 7**, although the first and second examples of operation respectively use SYNC signals and events exclusively, it is contemplated that the accelerator **44** can use a combination of both SYNC signals and events.

[115] **FIG. 8** is a block diagram of the accelerator **44** of **FIG. 3** including multiple groups **120** of multiple pipeline units **78** (**FIG. 4**) or **100** (**FIG. 5**) according to an embodiment of the invention. Including multiple groups **120** of pipeline units increases the functionality of the accelerator **44**, and allows a designer to increase the accelerator's efficiency by grouping pipeline units that perform related operations. For clarity, the accelerator **44** of **FIG. 8** is discussed as having multiple pipeline units **78**, it being understood that the accelerator may include multiple pipeline units **100** or a combination of units **78** and **100**. Furthermore, the pipeline units **78** do not include industry-standard bus interfaces **91** (this interface is external in this embodiment), although they may in another embodiment.

**[116]** The accelerator **44** includes six groups **120<sub>1</sub> – 120<sub>6</sub>** of pipeline units **78**, each group having three pipeline units and a respective intra-group communication-bus router **110<sub>1</sub> - 110<sub>6</sub>** that interconnects the pipeline units to each other and to the other pipeline-unit groups. Although the accelerator **44** is discussed as including six groups **120<sub>1</sub> - 120<sub>6</sub>** of three pipeline units **78** each, other implementations of the accelerator may include virtually any number of groups of any number pipeline units, and not all the groups need have the same number of pipeline units. Furthermore, the communication-bus routers **110<sub>1</sub>-110<sub>6</sub>** may be omitted as discussed above in conjunction with the accelerator **44** of **FIG. 6**.

**[117]** The pipeline-unit group **120<sub>1</sub>** includes three pipeline units **78<sub>1</sub> – 78<sub>3</sub>**, which are connected to an intra-group communication-bus router **110<sub>1</sub>** via branches **94<sub>1</sub>, 94<sub>2</sub>, and 94<sub>3</sub>**, respectively, of a communication bus in a manner similar to that discussed above in conjunction with **FIG. 7**. The other groups **120<sub>2</sub> – 120<sub>6</sub>** are similar.

**[118]** The communication-bus routers **110<sub>1</sub> - 110<sub>3</sub>** of the groups **120<sub>1</sub> - 120<sub>3</sub>** are connected to a first-level router **122<sub>1</sub>** via respective branches **124<sub>1</sub> – 124<sub>3</sub>** of a first-level bus **126<sub>1</sub>**. The router **122<sub>1</sub>** and bus **126<sub>1</sub>** allow the pipeline units **78<sub>1</sub> – 78<sub>9</sub>** to communicate with each other.

**[119]** Similarly, the communication-bus routers **110<sub>4</sub> - 110<sub>6</sub>** are connected to a first-level router **122<sub>2</sub>** via respective branches **128<sub>1</sub> – 128<sub>3</sub>** of a first-level bus **126<sub>2</sub>**. The router **122<sub>2</sub>** and bus **126<sub>2</sub>** allow the pipeline units **78<sub>10</sub> – 78<sub>18</sub>** to communicate with each other.

**[120]** The first-level routers **122<sub>1</sub>** and **122<sub>2</sub>** are connected to a second-level router **130** via respective branches **132<sub>1</sub> – 132<sub>2</sub>** of a second-level bus **134**. The router **130** and bus **134** allow the pipeline units **78<sub>1</sub> – 78<sub>18</sub>** to communicate with each other and with other peers/devices as discussed below.

**[121]** The pipeline bus **50** and a secondary pipeline bus **136** are coupled to the second-level router **130** via the respective industry-standard bus interfaces **91<sub>1</sub>** and **91<sub>2</sub>**. The secondary pipeline bus **136** may be connected to peers, such as the host processor **42** (**FIG. 3**), or peripherals, such as a hard-disk drive (not shown), that are not

coupled to the pipeline bus **50**. Furthermore, either or both of the busses **50** and **136** may be coupled to peers or peripherals via a network or the internet (neither shown) such that the accelerator **44** can be remotely located from other peers, such as the host processor **42** (**FIG. 3**).

5   **[122]**           A bus **138** directly connects one or more SYNC signals to all of the pipeline units **78<sub>1</sub> – 78<sub>18</sub>** and to other peers, such as the host processor **42** (**FIG. 3**), or devices (not shown).

10   **[123]**           Still referring to **FIG. 8**, in one example of operation, each group **120<sub>1</sub> – 120<sub>6</sub>** of pipeline units **78** processes data from a respective sensor of a sonar array (not shown) that is coupled to the secondary pipeline bus **136**. Because the pipeline units **78<sub>1</sub> – 78<sub>3</sub>** of the group **120<sub>1</sub>** are interconnected by a single router **110<sub>1</sub>**, these pipeline units can communicate with each other more quickly than they can with the pipeline units **78<sub>4</sub> – 78<sub>18</sub>** of the other groups **120<sub>2</sub> – 120<sub>6</sub>**. This higher communication speed is also present in each of the other groups **120<sub>2</sub> – 120<sub>6</sub>**. Consequently, a designer can increase the processing speed of the accelerator **44** by grouping together pipeline units that frequently transfer data or otherwise communicate among themselves.

15   **[124]**           In general, the pipeline units **78<sub>1</sub> – 78<sub>18</sub>** communicate with each other and with peers, such as the host processor **42** (**FIG. 3**), and devices coupled to the buses **50** and **136** in a manner similar to that discussed above in conjunction with **FIG. 7**. For example, a sensor (not shown) coupled to the bus **136** communicates with the pipeline unit **78<sub>1</sub>** via the industry-standard bus interface **91<sub>1</sub>**, the second-level router **130**, the first level router **122<sub>1</sub>**, and the intra-group router **110<sub>1</sub>**. Similarly, the pipeline unit **78<sub>1</sub>** communicates with the pipeline unit **78<sub>7</sub>** via the routers **110<sub>1</sub>**, **122<sub>1</sub>**, and **110<sub>3</sub>**, and communicates with the pipeline unit **78<sub>10</sub>** via the routers **110<sub>1</sub>**, **122<sub>1</sub>**, **130**, **122<sub>2</sub>**, and

20   **110<sub>4</sub>**.

25   **[125]**           The preceding discussion is presented to enable a person skilled in the art to make and use the invention. Various modifications to the embodiments will be readily apparent to those skilled in the art, and the generic principles herein may be applied to other embodiments and applications without departing from the spirit and

scope of the present invention. Thus, the present invention is not intended to be limited to the embodiments shown, but is to be accorded the widest scope consistent with the principles and features disclosed herein.